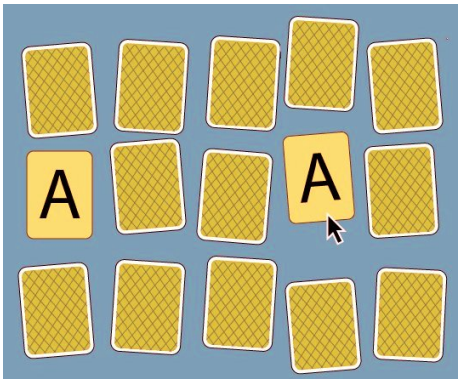
IOI 2010 WATERLOO ontario | canada
the 22nd International Olympiad in Informatics

Day 2 Task 1

Memory



A game called *Memory* is played using 50 cards. Each card has one of the letters from A to Y (ASCII 65 to 89) printed on the face, so that each letter appears on exactly two cards. The cards are shuffled into some random order and dealt face down on the table.

Jack plays the game by turning two cards face up so the letters are visible. For each of the 25 letters Jack gets a candy from his mother the first time he sees both copies of the letter on the two face up cards. For example, the first time Jack turns over both cards that contain the letter M, he gets a candy. Regardless of whether the letters were equal or not, Jack then turns both cards face down again. The process is repeated until Jack receives 25 candies – one for each letter.

You are to implement a procedure **play** that plays the game. Your implementation should call the procedure **faceup(C)** which is implemented by the grader. **C** is a number between 1 and 50 denoting a particular card you wish to be turned face up. The card must not currently be face up. **faceup(C)** returns the character that is printed on the card **C**.

After every second call to **faceup**, the grader automatically turns both cards face down again.

Your procedure **play** may only terminate once Jack has received all 25 candies. It is allowed to make calls to **faceup(C)** even after the moment when Jack gets the last candy.

Example.

The following is one possible sequence of calls your procedure **play** could make, with explanations.

Call	Returned value	Explanation
faceup(1)	'B'	Card 1 contains B.
faceup(7)	'X'	Card 7 contains X. The letters are not equal.
The grader automatically turns cards 1 and 7 face down.		
faceup(7)	'X'	Card 7 contains X.
faceup(15)	'O'	Card 15 contains O. The letters are not equal.
The grader automatically turns cards 7 and 15 face down.		
faceup(50)	'X'	Card 50 contains X.
faceup(7)	'X'	Card 7 contains X. Jack gets his first candy.
The grader automatically turns cards 50 and 7 face down.		
faceup(7)	'X'	Card 7 contains X.
faceup(50)	'X'	Card 50 contains X. Equal letters, but Jack gets no candy.
The grader automatically turns cards 7 and 50 face down.		
faceup(2)	'B'	Card 2 contains B.

...
(some function calls were omitted)

Day 2 Task 1

Memory

...

faceup(1)	'B'	Card 1 contains B.
faceup(2)	'B'	Card 2 contains B. Jack gets his 25th candy.

1 Subtask [50 points]. Implement any strategy that obeys the rules of the game and finishes it within the time limit.

For example, there is a simple strategy that always makes exactly $2*(49+48+...+2+1) = 2450$ calls to**faceup(C)**.

2 Subtask [50 taškų]. Implement a strategy that finishes any possible game with at most 100 calls to**faceup(C)**.

Implementation Details.

- Use the *RunC programming and test environment*
- Implementation folder: `/home/ioi2010-contestant/memory/` (prototype: memory.zip)
- To be implemented by contestant: `memory.c` or `memory.cpp` or `memory.pas`
- Contestant interface: `memory.h` or `memory.pas`
- Grader interface: `grader.h` or `graderlib.pas`
- Sample grader: `grader.c` or `grader.cpp` or `grader.pas` and graderlib.pas
- Sample grader input: `grader.in.1`
  Note: the input file contains one line with 50 characters denoting the letters on the cards, in order, from 1 to 50.
- Expected output for sample grader input: if your implementation is correct, the output file will contain `OK n` where n is the number of calls to `faceup(C)`.
- Compile and run (command line): `runc grader.c` or `runc grader.cpp` or `runc grader.pas`
- Compile and run (gedit plugin): *Control-R*, while editing any implementation file.
- Submit (command line): `submit grader.c` or `submit grader.cpp` or `submit grader.pas`
- Submit (gedit plugin): *Control-J*, while editing any implementation or grader file.